5. Control

In this chapter we will do:

- Feedback Control
- On/Off Controller
- PID Controller

Motor Control

Why use control at all?

- Correct or wrong?
  Supplying a certain voltage / pulse-width will make the motor spin at a certain speed.
- Wrong!
  This will only work in idle condition, not under load (e.g. driving up a slope)

Motor Control

Note:

- Time delay of controller
  (for measurement and calculation)
- Time delay of motor
  (physical reaction time of actuator in real world)
- Inaccuracy of encoder data
- Inaccuracy of actuator movement

How is this done?

1. Read current speed
2. Compare with desired speed
3. Change motor output accordingly
5.1 On-Off Controller

Simplest case: Only on/off of control value “Bang-Bang Controller”

Control signal accelerates actuator
- Without control signal, actuator slows down
- May add hysteresis to avoid quick oscillation

Bang-Bang-Controller (on-off controller)
\[
R(t) = \begin{cases} 
K_C & \text{if } v_{act}(t) < v_{des}(t) \\
0 & \text{else} 
\end{cases}
\]

On-Off Controller with Hysteresis

Note:
- Update is not continuous
- Update is e.g. every 10ms

\[
R(t+\Delta t) = \begin{cases} 
K_C & \text{if } v_{act}(t) < v_{on}(t) \\
0 & \text{if } v_{act}(t) > v_{off}(t) \\
R(t) & \text{else} 
\end{cases}
\]
On-Off Controller with Hysteresis

\[
R(t+\Delta t) = \begin{cases} 
K_C & \text{if } v_{act}(t) < v_{on}(t) \\
0 & \text{if } v_{act}(t) > v_{off}(t) \\
R(t) & \text{else}
\end{cases}
\]

On-Off Controller

**Bang-Bang-Controller (on-off controller)**

\[
R(t) = \begin{cases} 
K_C & \text{if } v_{act}(t) < v_{des}(t) \\
0 & \text{else}
\end{cases}
\]

**From THEORY to PRACTICE**

**1. Write a control subroutine**

a. Read encoder data (INPUT)
b. Compute new output value R(t)
c. Set motor speed (OUTPUT)

2. Call control subroutine periodically

   e.g. every 1/100 s

int `main()`
{
    /* motor control */
    ???
    return 0;
}

See library.html:

int `QUADRead` (QuadHandle handle);
Input: (handle) ONE decoder-handle
Output: 32bit counter-value (-2^31 .. 2^31-1)
Semantics: Read actual Quadrature-Decoder counter, initially zero.
Note: A wrong handle will ALSO result in a 0 counter value!!

int `MOTORDrive` (MotorHandle handle, int speed);
Input: (handle) logical-or of all MotorHandles which should be driven
(speed) motor speed in percent
Valid values: -100 - 100 (full backward to full forward)
0 for full stop
Output: (return code) 0 = ok
-1 = error wrong handle
Semantics: Set the given motors to the same given speed
On-Off Controller

From THEORY to PRACTICE
1. Write a control subroutine
   a. Read encoder data (INPUT)
   b. Compute new output value \( R(t) \)
   c. Set motor speed (OUTPUT)

```
void onoff_controller()
{
  int enc_new, v_act, r_mot, err;
  
  enc_new = QUADRead(enc1);
  v_act = (enc_new - enc_old) * 100;
  if (v_act < v_des) r_mot = Kc;
  else  r_mot = 0;

  err = MOTORDrive(mot1, r_mot);
  if (err) printf("error: motor");

  enc_old = enc_new;
}
```

- speed in “encoder-ticks per s” assuming 100 calls per s
- variable needs to keep value for next call of routine

On-Off Controller

Subsequent encoder readings (example):
- \( FFF8 \)
- \( FFFE \) (+6)
- \( 0002 \) (+4)

Encoder “Overflow”
- our program computes: -FFFFC
- which equals (2’s complement): +0004
- similar for underflow

• So far, so good!

• However, is there a problem with overflow/underflow ??
On-Off Controller

From THEORY to PRACTICE

2. Call control subroutine periodically
c.e. every 1/100 s

See library.html:

TimerHandle OSAttachTimer(int scale, TimerFnc function);
Input: (scale) prescale value for 100Hz Timer (1 to ...)
(TimerFnc) function to be called periodically
Output: (TimerHandle) handle to reference the IRQ-slot
Semantics: Attach a irq-routine (void function(void)) to the irq-list.
The scale parameter adjusts the call frequency (100/scale Hz)
of this routine to allow many different applications.

int OSDetachTimer(TimerHandle handle)
Input: (handle) handle of a previous installed timer irq
Output: 0 = handle not valid
1 = function successfully removed from timer irq list
Semantics: Detach a previously installed irq-routine from the irq-list.

int main()
{ TimerHandle t1;
  t1 = OSAttachTimer(1, onoff_controller);
  while (1) /* endless loop - never returns */
  { /* other tasks or idle */

    OSDetachTimer(t1); /* not used */
    return 0; /* not used */
  }

  t1 = OSAttachTimer(1, onoff_controller);
  while (KEYRead() != KEY4) /* check end key */
  { /* set desired speed with input keys */
    OSDetachTimer(t1);
    return 0;
  }
}

better for our purpose
5.2 Proportional Controller

P-controller (proportional controller)

\[ R(t) = K_p \cdot (v_{\text{des}}(t) - v_{\text{act}}(t)) \]

Linear change of control value to reduce error → better controller

Proportional Controller

\[ R(t) = K_p \cdot e(t) \quad \text{with} \quad e(t) = v_{\text{des}}(t) - v_{\text{act}}(t) \quad \text{“error function”} \]

- \( K_p \) is a constant value, “controller gain”
- \( K_p \) must be selected to achieve fast goal speed without overshoot
- Note: No equilibrium when desired velocity is reached:
  if \( v_{\text{des}}(t) = v_{\text{act}}(t) \) then \( R(t) = 0 \)

Implementation in C:

```c
    e_func = v_des - v_act;
    r_mot  = Kp*e_func;
```
5.3 Integral Controller

Adding I-Controller (Integral Controller) Part

- **Problem:** P-Controller may reach equilibrium without reaching the target velocity → steady state error
- **Solution:** Integral part is used to eliminate steady state error

\[ R(t) = K_p \cdot \left[ e(t) + \frac{1}{T_I} \int_0^t e(t) \, dt \right] \]

\[ R(t) = K_p \cdot e(t) + K_I \cdot \int_0^t e(t) \, dt \]

Implementation - Naïve approach:
- integral corresponds to SUM
- use ARRAY to store e.g. 10 last values and sum up

Implementation in C:
- Use **trapezoid** rule

\[ R_n - R_{n-1} = K_p \cdot (e_n - e_{n-1}) + K_I \cdot Q_I \cdot \sum_{i=0}^{n} \frac{e_i + e_{i-1}}{2} \]

\[ R_n = R_{n-1} + K_p \cdot (e_n - e_{n-1}) + K_I \cdot \frac{(e_n + e_{n-1})}{2} \]
### Integral Controller

\[ R(t) = K_p \cdot e(t) + K_I \int e(\tau) d\tau \]

```c
static int r_old=0, e_old=0;
...
  e_func = v_des - v_act;
  r_mot = r_old + Kp*(e_func-e_old) + Ki*(e_func+e_old)/2;
  r_mot = min(r_mot, +100); /* limit output */
  r_mot = max(r_mot, -100); /* limit output */
  r_old = r_mot;
  e_old  = e_func;
```

### 5.4 Derivative Controller

**Adding D-Controller (Derivative Controller) Part**

- **Problems:** P-Controller responds slow to change in input
  P-Controller with high gain tends to oscillate
- **Solution:** Add a derivative term for response/damping

\[ R(t) = K_p \cdot [ e(t) + \frac{1}{T_I} \int e(\tau) d\tau + T_D \frac{de(t)}{dt} ] \]

or equivalent with \( K_I = \frac{K_p}{T_I} \) and \( K_D = K_p \cdot T_D \):

\[ R(t) = K_p \cdot e(t) + K_I \int e(\tau) d\tau + K_D \frac{de(t)}{dt} \]

**Implementation in C:**

- Derivative corresponds to **DIFFERENCE**
5.5 PID Controller

Full PID Solution:
\[ R(t) = K_p \cdot e(t) + K_i \int_0^t e(t) \, dt + K_d \frac{de(t)}{dt} \]

\[ R_n = K_p \cdot e_n + Q_i \cdot \left( \frac{1}{t_{delta}} \sum_{i=0}^{n} \frac{e_i + e_{i-1}}{2} \right) + Q_D \cdot \left( \frac{e_n - 2e_{n-1} + e_{n-2}}{t_{delta}} \right) \]

\[ R_n - R_{n-1} = K_p \cdot (e_n - e_{n-1}) + K_i \cdot (e_n + e_{n-1})/2 + K_D \cdot (e_n - 2e_{n-1} + e_{n-2}) \]

PID Controller

**PID-Controller** is combination of P, I, and D controller

- Simple, universal controller
- Trade-offs: Response time vs. stability
- Noise limits max. proportional gain
- Note physical limits of actuator e.g. max acceleration, velocity

```c
static int r_old=0, e_old=0, e_old2=0;
...
e_func = v_des - v_act;
r_mot  = r_old + Kp*(e_func-e_old) + Ki*(e_func+e_old)/2 + Kd*(e_func - 2* e_old + e_old2);
r_mot = min(r_mot, +100); /* limit output */
r_mot = max(r_mot, -100); /* limit output */
e_old2 = e_old;
e_old  = e_func;
r_old = r_mot;
```
Tuning: Find suitable parameter settings for PID controller
1. Select typical operating setting for desired speed, turn off integral and derivative part, then increase $K_p$ to max. or until oscillation occurs.
2. If system oscillates, divide $K_p$ by 2.
3. Increase $K_D$ and observe behavior when changing desired speed by about 5%. Choose a value of $K_D$ which gives a fast damped response.
4. Slowly increase $K_I$ until oscillation starts. Then divide $K_I$ by 2 or 3.

Problems:
- What happens if desired speed > physical max. speed?
- Error always > 0 $\Rightarrow$ integral term accumulates
- If the desired speed is reached for a possible speed, the integral residue will make the controller overshoot

Solutions:
- Use max. and min. values for integral
- Stop I-summation upon saturation
- Use more sophisticated velocity algorithms or analytical methods

5.6 Velocity Control vs. Position Control
- “Velocity Control” is what we have achieved so far
  We can make a motor drive at a certain speed and make it maintain that speed
- “Position Control” also incorporates a start/stop phase and makes the motor stop at a certain location
  E.g. exactly after one full revolution
  This is important for many applications, e.g. driving a vehicle

Acknowledgements
Baltes: PID formulas and tuning
Kasper: PID formulas
Braunschädel: PID measurements
Velocity Control vs. Position Control

Example for position control:
move object from $s_0$ to $s_1$

- Simple assumption:
  - Constant acceleration / deceleration
  - Linear velocity in start/stop phase
  - Quadratic distance in start/stop phase
  - Linear otherwise

A practical way of achieving position control
- Use velocity control in start phase and continuous phase
- Monitor current position and calculate begin of end phase
- Continuously update speed while decelerating
5.7 Driving a Vehicle

Two independently driven wheels
Task 1: Straight line

How can you synchronize the motors?
1. Use one PID controller per wheel
   • pros / cons?
   • how can you avoid irregularities?
     (each wheel is controlled but slight disturbances will result in curve)

How can you synchronize the motors?
1. Use one PID controller per wheel
   • pros / cons?
   • how can you avoid irregularities?
     (each wheel is controlled but slight disturbances will result in curve)

Task 2: Curve
• Adapt structure for straight line with offset for both wheels to set path curvature
Driving a Vehicle

Other solutions are possible as well!
E.g. Use one PID controller for speed
     and one PID controller for straightness
     • pros / cons?